
课程练习


5.1 类的复用：组合与继承

第5章 类的继承

5.1 类的复用：组合与继承

“复用”也被称作“重用”，是重复使用的意思，即将已有的软件元素使用在新的软件开发中。这里所说的“软件元素”包括程序代码、测试用例、设计文档、设计过程、需求分析文档甚至领域知识和经验等。通常，可重用的元素称作软构件。构件的大小称为构件的粒度，可重用的软构件越大，重用的粒度越大。使用软件重用技术可以减少软件开发活动中大量的重复性工作，这样就能提高软件的生产率，降低开发成本，缩短开发周期。由于软构件大多经过严格的质量认证，并在实际运行环境中得到校验，重用软构件还有助于改善软件质量。

一般来说，软件重用可分为如下 3 个层次。

（1）知识重用（例如，软件工程知识的重用）。

（2）方法和标准的重用（例如，面向对象方法或国家制定的软件开发规范的重用）。

（3）软件成分和架构的重用。

Java 提供了实现代码重用的两种方式：组合以及继承。

- 组合：新的类由现有类的对象所组成。
- 继承：按照现有类的类型派生出新类。

5.1.1 类的组合

如果一个类把另外一个类的对象作为自己的成员变量，即内嵌其他类的对象作为自己的成员，称为类的组合。类的组合是实现软件重用的一种重要方式。组合表示类的对象之间是“has-a”（有一个）的包含关系，即一类对象包含另一类对象，如，A house has a room。

例如，需要计算圆柱的体积，计算公式如下：

柱的体积＝底面积×高

如果一个类过于复杂，可以将其拆分成多个类，拆分成的类成为组合类的子对象。例如，一个完整圆柱由底面和高组成，可以将底面的圆类拆分出去，通过在圆柱类中声明一个圆类对象，将两个类组合起来。

1. 关联关系的 UML 建模

圆类对象作为圆柱类的一个成员，两者的关系是关联关系，就是圆柱类（Pillar）关联于圆类（Circle）。图 5.1 是 Pillar 类和 Circle 类关联关系的 UML 描述。

如果 A 类中的成员变量是用 B 类（接口）声明的对象，那么 A 和 B 的关系是关联关系，称 A 类的对象关联于 B 类的对象或 A 类的对象组合了 B 类的对象。如果 A 关联于 B，那么可以通过一条实线连接 A 和 B 的 UML 类图，实线的起始端是 A 的 UML 图，终点端是 B 的 UML 图，但终点端使用一个指向 B 的 UML 图的方向箭头表示实线的结束。

2. 类组合实现复用的代码设计

【代码 5-1】 类的组合。

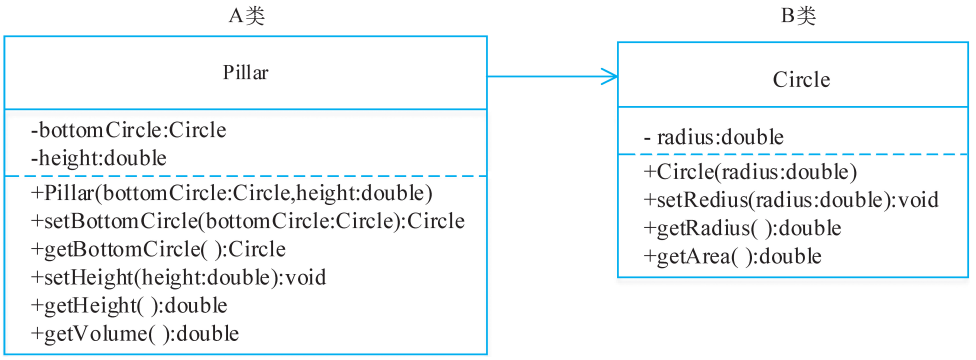

图 5.1　Pillar 类和 Circle 类关联关系的 UML 描述

Circle.java

```
1  public class Circle {
2      /** 半径 */
3      private double radius;
4
5      public Circle(double radius) {
6          this.radius =radius;
7      }
8
9      public double getRadius() {
10         return radius;
11     }
12
13     public void setRadius(double radius) {
14         this.radius =radius;
15     }
16
17     /** 计算面积 */
18     public double getArea() {
19         return Math.PI * radius * radius;
20     }
21 }
```

Pillar.java

```
1  public class Pillar {
2      /** 圆柱的底面对象 */
3      private Circle bottomCircle;
4      /** 圆柱的高 */
5      private double height;
6
7      /** 将 Circle 类对象作为 Pillar 类构造器的参数 */
8      public Pillar(Circle bottomCircle, double height) {
9          this.bottomCircle =bottomCircle;
10         this.height =height;
11     }
12
13     public Circle getBottomCircle () {
14         return bottomCircle;
15     }
16
17     public void setBottomCircle (Circle bottomCircle) {
18         this.bottomCircle =bottomCircle;
```

```
    }

    public double getHeight() {
        return height;
    }

    public void setHeight(double height) {
        this.height =height;
    }

    /** 计算体积 */
    public double getVolume() {
        return bottomCircle.getArea() * height;
    }
}
```

TestPillar.java

```
public class TestPillar {
    public static void main(String[] args) {
        /** 创建 Circle 对象 */
        Circle bottomCircle =new Circle(1.5);
        /** 利用已有的 Circle 对象创建 Pillar 对象,以实现对象组合 */
        Pillar pillar =new Pillar(bottomCircle, 5.0);
        System.out.println("圆柱的体积为: " +pillar.getVolume());
    }
}
```

代码 5-1 中定义 Pillar 类时使用了 Circle 类对象作为其成员，意味着一个 Pillar 对象包含一个 Circle 对象。这样就允许在新类(Pillar 类)中直接复用旧类(Circle 类)的 public 方法(如 getArea()方法)。在使用时通过在 Pillar 类的构造器中传入 Circle 类对象来实现对象的组合。

通过以上实例可以看出，组合就是把旧类(Circle 类)对象作为新类的成员变量组合进来，用以实现新类(Pillar 类)的功能，用户看到的是新类(Pillar 类)的方法(如 getVolume()方法)，而不能看到被组合对象的方法(如 getArea()方法)。因此，通常需要在新类里使用 private 修饰被组合的旧类对象。利用组合来实现复用，是复用现有代码的功能，而非它的形式。

如果要计算圆锥的体积，则可继续将 Circle 对象作为圆锥(Cone)类的成员变量，达到多次复用的目的。

5.1.2 类的继承

如果需要复用一个类，除了把该类当成另一个类的组合成分外，还可以把这个类当成基类来继承进而派生出新的类。不管是组合还是继承，都允许在新类(对于继承是子类)中直接复用旧类的方法。利用继承实现复用，是在不改变现有类的基础上，复用现有类的形式并在其中添加新代码。

继承是所有 OOP(面向对象的编程)语言，包括 Java 语言不可缺少的组成部分。面向对象程序设计的核心是定义类。前面介绍的类是直接定义的，除此之外，还可以在已经定义类的基础上定义新的类，由新的类继承已经定义类的部分代码实现部分代码的重用。本章通过学生和研究生之间存在的继承，也称泛化关系，讨论一般继承(泛化)关系的 Java 描述以及所引出的有关问题。

下面以从学生类派生出研究生类为例，介绍类继承的设计方法。

1. 派生关系的 UML 建模

研究生也是学生，即研究生是学生的一部分，学生是研究生的抽象。这种关系在面向对象的程序设计中用继承表示。对于本例，可以说是 Student 类派生出 GradStudent 类，也可以说 GradStudent 类继承了 Student 类。这种基类和派生类的关系称为 is-a（是一个、是一种）关系，例如，研究生 is-a 学生。

图 5.2 是 GradStudent 类和 Student 类继承关系的 UML 描述。如果一个类是另一个类的子类，那么 UML 通过使用一条实线连接两个类的 UML 图来表示两者之间的继承关系，实线的起始端是子类的 UML 图，终点端是父类的 UML 图，但终点端使用一个空心的三角形表示实线的结束。

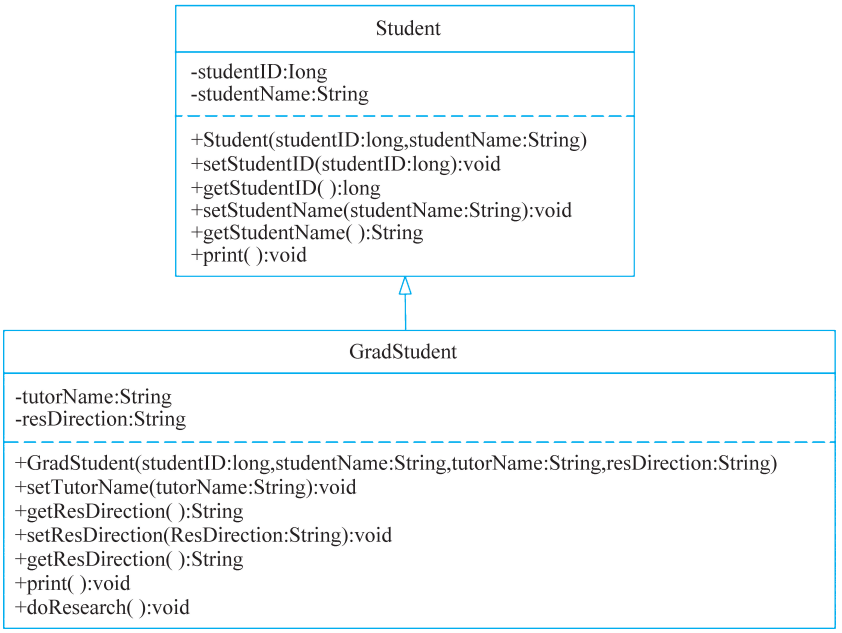

图 5.2　类的继承关系

2. 由 Student 类派生 GradStudent 类的代码设计

Java 类继承的语法格式为：

```
[修饰符] class 子类名称 extends 父类名{
}
```

修饰符：可选，用于指定类的访问权限，可选值为 public、abstract 和 final。

【代码 5-2】　GradStudent 类继承 Student 类的 Java 代码。

Student.java

```
1   /** 学生类 */
2   public class Student {
3       /** 学号 */
4       private long studentID;
5       /** 学生姓名，使用了 protected */
6       protected String studentName;
```

```
7
8       /** 构造器 */
9       public Student(long studentID, String studentName) {
10          this.studentID = studentID;
11          this.studentName = studentName;
12      }
13
14      public long getStudentID() {
15          return studentID;
16      }
17
18      public void setStudentID(long studentID) {
19          this.studentID = studentID;
20      }
21
22      public String getStudentName() {
23          return studentName;
24      }
25
26      public void setStudentName(String studentName) {
27          this.studentName = studentName;
28      }
29
30      /** 输出信息 */
31      public void print() {
32          System.out.println("学号: " + studentID);
33          System.out.println("姓名: " + studentName);
34      }
35  }
```

GradStudent.java

```
1   /** 研究生类 */
2   public class GradStudent extends Student {
3       /** 导师姓名 */
4       private String tutorName;
5       /** 研究方向 */
6       private String resDirection;
7
8       /** 构造器 */
9        public GradStudent(long studentID, String studentName, String tutorName, String
        resDirection) {
10          //调用父类构造方法
11          super(studentID, studentName);
12          this.tutorName = tutorName;
13          this.resDirection = resDirection;
14      }
15
16      public String getTutorName() {
17          return tutorName;
18      }
19
20      public void setTutorName(String tutorName) {
21          this.tutorName = tutorName;
22      }
23
24      public String getResDirection() {
25          return resDirection;
26      }
27
28      public void setResDirection(String resDirection) {
```

```
29          this.resDirection =resDirection;
30      }
31
32      /** 输出信息 */
33      public void print() {
34          System.out.println("学号: " +this.getStudentID());
35          System.out.println("姓名: " +studentName);
36          System.out.println("导师姓名: " +tutorName);
37          System.out.println("研究方向: " +resDirection);
38      }
39
40      public void doResearch() {
41          System.out.println(this.getStudentName() +" is doing research");
42      }
43  }
```

TestExtends.java

```
1   /** 测试类 */
2   public class TestExtends {
3       /** 主方法 */
4       public static void main(String[] args) {
5           Student st =new Student(123456, "王舞");
6           st.print();
7           GradStudent gs =new GradStudent(654321, "李司", "张伞", "人工智能");
8           gs.print();
9       }
10  }
```

程序运行结果：

```
学号: 123456
姓名: 王舞
学号: 654321
姓名: 李司
导师姓名: 张伞
研究方向: 人工智能
```

说明：

(1) 关键字 extends 表示扩展或派生，即以一个类为基础派生出一个新类。这个新生成的类称为派生类或直接子类；原始的类作为派生类形成的基础存在，称为基类，也称为派生类的超类或父类。在本例中，class GradStudent extends Student 表明 GradStudent 类是以 Student 为基类扩展而成的派生类。派生类也可以继续扩展成新的派生类。

从另一方面看，extends 关键字使派生类继承了基类的属性和方法，因此派生类无法脱离基类而存在。

(2) 子类会继承父类的所有属性和方法(静态成员、构造方法除外)。但是，对于父类的私有属性和方法，子类是无法访问的(只是拥有，但不能使用)。虽然子类不能直接访问从父类继承过来的私有属性，但是可以通过父类提供的 public 访问器间接访问。从父类继承过来的成员，会保持其原有的访问权限和功能，它可以被子类中自己定义的任何实例方法使用。

(3) 父类中的静态成员虽然未被子类继承，但可以通过子类名或子类对象访问它们。可以认为，父类的静态成员被父类及其所有子类对象所共享。

(4) 注意，在类 Student 中成员 studentName 改用 protected 修饰，而不是用 private 修

饰。因为 private 将所修饰的成员的访问权限限制在本类中(即不允许子类直接访问父类的 private 成员),而 protected 允许将所修饰成员的访问权限扩展到派生类中。这样,父类的 studentName 成员才能被子类 GradStudent 中的 print()方法直接访问(当然,也可用 getStudentName()方法访问),见 GradStudent 类代码的第 35 行。子类 GradStudent 中的 print()方法通过调用父类提供的 getStudentID()来访问父类的 private 成员 studentID,见 GradStudent 类代码的第 34 行。一般情况下,父类数据成员的访问权限都设置为 private,是否要设置成 protected,视具体情况而定。这里将 studentName 的修饰符设为 protected,只是为了说明子类对父类成员的访问情况。

(5) 继承表明了两个类之间的父子关系,让父类和子类之间建立起了联系,子类自动拥有父类的成员(静态成员、构造方法除外),包括成员变量和成员方法,使父类成员得以传承和延续(如本例中的 studentName 和 studentID);子类可以重新定义(重写)父类的成员,使父类成员适应新的需求(例如,子类 GradStudent 的 print()方法对父类的 print()方法进行了重写/覆盖,方法覆盖的内容在后续章节介绍);子类也可以添加新的成员,使类的功能得以扩充(例如,子类 GradStudent 新增了数据成员 tutorName、resDirection,新增了方法成员 doResearch())。但是,子类不能删除父类的成员。

(6) 利用继承,可以先编写一个具有共有属性和方法的父类(如 Student 类),根据该父类再编写具有特殊属性和方法的子类(如 GradStudent 类),子类继承父类的状态和行为,并根据需要增加它自己的新的状态和行为,也可以修改从父类继承过来的行为。本科生也是学生,这样,可以在 Student 类的基础上再派生本科生 Undergraduate 类,代码如下。

```
1  /** 本科生类 */
2  public class Undergraduate extends Student {
3      /** 专业名称 */
4      private String major;
5
6      /** 带参构造器 */
7      public Undergraduate(long studentID, String studentName, String major) {
8          super(studentID, studentName);
9          this.major =major;
10     }
11
12     public String getMajor() {
13         return major;
14     }
15
16     public void setMajor(String major) {
17         this.major =major;
18     }
19
20     public void print() {
21         super.print();
22         System.out.println("专业名称: " +major);
23     }
24 }
```

目前,学生类派生出了本科生类和研究生类,学生类是本科生类和研究生类的共同父类。

3.Java 继承规则

Java 语言的继承有如下特征。

(1) 每个子类只能有一个直接父类(不允许多重继承),但一个父类可以派生出多个子类,见图 5.3(a)。

(2) 派生具有传递性。Java 允许多层继承,如果类 A 派生了类 B,类 B 又派生了类 C,则 C 不仅继承了 B,也继承了 A,见图 5.3(b)。这样,一个类就能拥有多个间接父类。

(3) 不可循环派生。若 A 派生了类 B,类 B 又派生了类 C,则类 C 不可派生 A,见图 5.3(c)。

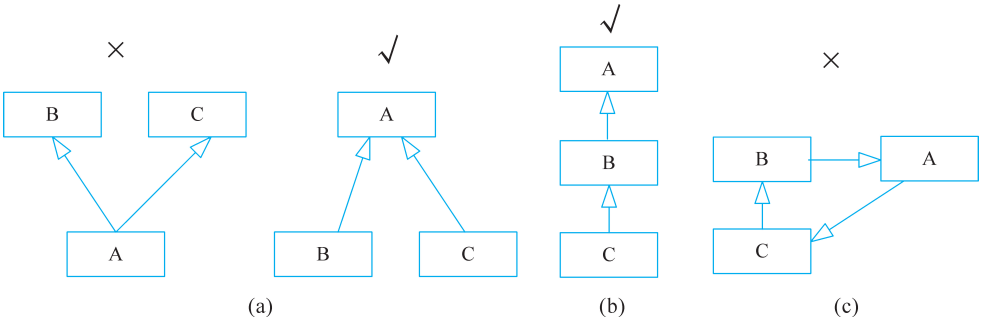

图 5.3 Java 继承规则

(4) 组合优先。就是能使用组合就尽量使用组合,而不使用继承。

5.1.3 super 关键字

super 是 Java 的一个关键字,它有两种用法。

1. 用 super 调用父类(对象)的可见成员

可用 super 访问父类的可见数据成员。例如,GradStudent 类的 print()方法访问 studentName 也可用 super.studentName 进行访问。

【代码 5-3】 GradStudent 类的 print()方法用 super 访问父类的可见数据成员。

```
1  public void print() {
2      System.out.println("学号: " +this.getStudentID());
3      System.out.println("姓名: " +super.studentName);
4      System.out.println("导师姓名: " +tutorName);
5      System.out.println("研究方向: " +resDirection);
6  }
```

可用 super 调用父类的可见方法成员。调用父类方法的格式为:

```
super.方法名(参数)
```

例如,可将 GradStudent 类的 print()方法进行如下改造,用 super.print()调用父类的 print()方法。

【代码 5-4】 GradStudent 类的 print()方法用 super 调用父类的可见方法成员。

```
1  public void print() {
2      //用 super 调用父类的方法
3      super.print();
4      System.out.println("导师姓名: " +tutorName);
5      System.out.println("研究方向: " +resDirection);
6  }
```

2. 用 super()代表父类构造器

父类的构造方法不能被子类继承,它们被显式或隐式地调用。使用 super 关键字显式

调用父类的构造方法。与 this()一样,它必须放在调用函数中的第 1 行,即当调用派生类的构造器实例化时首先要调用基类的构造器对从基类继承的成员进行实例化,再对本类新增成员进行实例化。调用父类构造方法的格式为:

```
super()或者 super(参数)
```

注意:要调用父类构造方法就必须使用关键字 super,而且这个调用必须是构造方法的第一条语句。

代码 5-2 中 GradStudent 类的第 11 行就是用 super 显式调用了父类的构造方法,如下:

```
//调用父类构造方法
super(studentID, studentName);
```

注意:

(1) 与 this()不同,使用 super()必须为所调用成员的访问权限允许,否则无法调用。

(2) this()和 super()不能同时存在,因为都要在第一行。

5.1.4 继承关系下的构造方法调用

在 Java 中构造一个类的实例时,将会调用沿着继承链的所有父类的构造方法。当构造一个子类的对象时,子类构造方法会在完成自己的任务之前,首先调用它的父类的构造方法。如果父类继承自其他类,那么父类构造方法又会在完成自己的任务之前,调用它自己的父类的构造方法。这个过程持续到沿着这个继承体系结构的最后一个构造方法被调用为止。这就是构造方法链。

如果父类没有定义构造方法,则调用编译器自动创建的不带参数的默认构造方法。如果父类定义了 public 的无参的构造方法,则在调用子类的构造方法前会自动先调用该无参的构造方法。如果父类只有有参的构造方法,没有无参的构造方法,则子类必须在构造方法中显式调用 super(参数列表)来指定某个有参的构造方法。如果父类定义有无参的构造方法,但无参的构造方法声明为 private,则子类同样必须在构造方法中显式调用 super(参数列表)来指定某个有参的构造方法。如果父类没有其他的有参构造方法,则子类无法创建。

如果没有显式地调用父类构造方法,编译器会自动地将 super()作为子类构造方法的第一条语句。例如,代码 5-5 中 Student 类的无参构造器 24 行之前,编译器会自动加上一条语句 super();。但是,其有参构造器的第 30 行之前就不会自动加上此条语句。因为 super()与 this()不能同时存在。所以,以下两段代码等价。

```
public Student() {
    System.out.println("(2) Student 无参构造器");
}
```

等价于:

```
public Student() {
    super();
    System.out.println("(2) Student 无参构造器");
}
```

代码 5-5 是 Student-GradStudent 类层次中的构造方法链演示。代码较之前有所精简，为了说明问题引入了另一个类 Person，让 Student 类继承自 Person 类。

【代码 5-5】 在 Student-GradStudent 类层次中的构造方法链。

```
/** 测试类 */
public class TestExtends {
    /** 主方法 */
    public static void main(String[] args) {
        GradStudent gs = new GradStudent(654321, "李司", "张伞", "人工智能");
    }
}

/** Person 类 */
class Person {
    /** 无参构造器 */
    public Person() {
        System.out.println("(1) Person 无参构造器");
    }
}

/** 学生类 */
class Student extends Person {
    private long studentID;
    private String studentName;

    /** 无参构造器 */
    public Student() {
        System.out.println("(2) Student 无参构造器");
    }

    /** 带参构造器 */
    public Student(long studentID, String studentName) {
        //调用本类另一构造器
        this();
        this.studentID = studentID;
        this.studentName = studentName;
        System.out.println("(3) Student 带参构造器");
    }
}

/** 研究生类 */
class GradStudent extends Student {
    private String tutorName;
    private String resDirection;

    /** 带参构造器 1 */
    public GradStudent(long studentID, String studentName) {
        //调用父类构造方法
        super(studentID, studentName);
        System.out.println("(4) GradStudent 带参构造器 1");
    }

    /** 带参构造器 2 */
     public GradStudent(long studentID, String studentName, String tutorName, String
     resDirection) {
        //调用本类类另一构造器
        this(studentID, studentName);
        this.tutorName = tutorName;
```